DEVSISTERS

# 아마존 클라우드와 함께한 1개월

## 쿠키런 사례중심

# 강연자 소개

DEVSISTERS

홍성진 (sungjinhong@devsisters.com)

- KAIST 전산과 학사
- 소프트웨어 엔지니어링 인턴, 구글 코리아
  - 통계기계번역팀, 한-영 번역 품질 향상: http://translate.google.com
- 시니어 소프트웨어 엔지니어, 위클레이
  - 영상/음성 통화 앱 시드 클라이언트/서버 개발: http://seed.weclay.com
- 테크니컬 리드, 데브시스터즈
  - 쿠키런 for Kakao, Ovenbreak 2 서버 개발

# 아마존 웹서비스

AWS를 사용하게 된 계기

# 2013년 초 데브시스터즈

DEVSISTERS

- 서버 개발자 2명, 시스템 엔지니어 0명
  - 한 분은 쿠키런 출시 전 전직 예정! **혼자 개발/관리해야함**
- 2013년 초 Ovenbreak 2 출시로 일정 성과를 냄
  - 한 명이 한 달만에 구현한 서버
  - 북미 AWS 를 사용, 현지 서버가 필요했으니 당연한 선택
- 쿠키런은 서버는 어떻게 할 것인가?
  - 서버는 어떻게 구현? Ovenbreak 2 서버를 재활용!
  - 한국에 서비스하려면 서버는 어디에 둘까?

# AWS 선택의 이유

DEVSISTERS

- 애로사항이 꽃피는 IDC 입주 및 서버구입
  - IDC는 어디로? 서버는 어느 밴더?
  - BMT, 서버 설치, 네트워크 구축할 시간과 인력 부족
  - 서버 확충에 대한 늦은 리드 타임 (최소 2주)
  - 그렇다고 많이 구입을 하자니 높은 초기 투자비용 필요
- 한국 클라우드 밴더들에대한 신뢰성 문제
  - 높은 트래픽 환경에서 제대로된 성능을 내지 못하는 사례 (LB 장애?!)
  - 상대적으로 기능이 부족하고 운영 경험 및 노하우 부족

# AWS 선택의 이유 (2)

DEVSISTERS

- 회사가 이미 AWS를 사용
- 서버 개발자 한명이 관리할 수 있는 서버 환경
    - API 및 원격으로 모든 시스템 관리 가능
- 단순 IaaS 라고 볼 수 없는 다양한 서비스 제공
    - EBS, AMI, Auto Scaling, ELB, S3, Route53, CloudFront, RDS, ...
    - 모든 서비스는 API로 제어가 가능
- AWS의 다양한 서비스와 조합하면 훨씬 더 큰 부가가치를 얻을 수 있을것
    - 특히 Auto Scaling, ELB, S3 등이 매력적

# 아마존 웹서비스

AWS의 간단한 소개

# AWS의 간단 소개

DEVSISTERS

- 웹 서비스 (Web Service) - 인프라(컴퓨터, 스토리지 등)을 객체처럼 다룬다
    - AMI (Amazon Machine Image) - 디스크 스냅샷; Class
    - 인스턴스 (Instance) - AMI를 부팅한 가상머신; Object(Instance) of AMI
    - ELB (Elastic Load Balancer) - 로드밸런서 as Object
    - EBS (Elastic Block Store) - 추상화된 NAS 디스크; Disk as Object
    - S3 (Simple Storage Service) - 추상화된 파일 스토리지; File as Object
- 모든 서비스는 API를 통해서 다룰 수 있다
    - 요즘 대부분이 사용하는 Web Console은 원래 없었음

# AWS S3 예제

DEVSISTERS

```
>>> import boto
>>> import time
>>> s3 = boto.connect_s3()

# Create a new bucket. Buckets must have a globally unique name
>>> bucket = s3.create_bucket('kgc-demo')

# Create a new key/value pair.
>>> key = bucket.new_key('mykey')
>>> key.set_contents_from_string("Hello World!")

# Sleep to ensure the data is eventually there.
>>> time.sleep(2)

# Retrieve the contents of ``mykey``.
>>> print key.get_contents_as_string()
'Hello World!'

# $ curl http://kgc-demo.s3.amazonaws.com/mykey
# Hello World!
```

# 쿠키런의 서버구조

출시 때 준비한 서버 구조 소개

# 출시 전 준비사항

DEVSISTERS

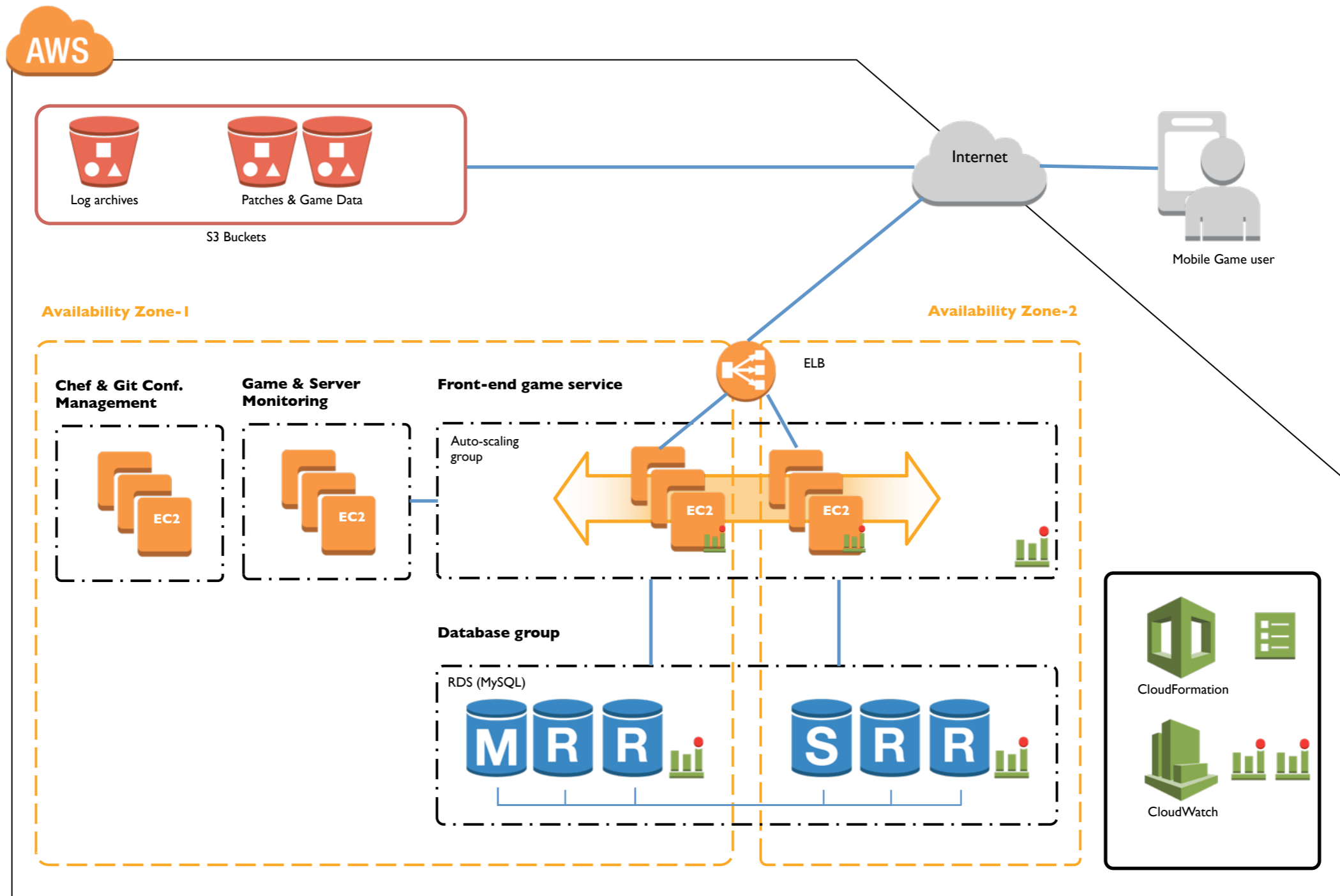
- 핵심 주안점: **최대한 자동화된 서버 인프라**
- CloudFormation, Chef, Git을 통하여 서버의 설정을 소스코드처럼 관리하자
  - 서버 구조 문서화 및 배포를 따로 **x**
- ELB, Auto Scaling을 꼭 써보자: 손으로 일일이 서버를 증가시키기 **x**
- S3를 사용하여 게임 데이터를 제공하자: 웹서버 구축 및 파일 호스팅 관리 **x**
- 모니터링은 확실하게 하자: 직접 서비스 모니터링 **x**

# 초기 소프트웨어 스택

DEVSISTERS

- 메인 API 서버
  - Java, Spring MVC, Mybatis, MySQL 5.5
  - Tomcat 6.0, Apache 2.2 (mod_jk), Ubuntu Linux 13.04
- GMS, 게임 데이터 관리 서비스
  - Python, Django, Boto
- 서비스 모니터링
  - CloudWatch, SNS, Zabbix, Statsd, Graphite

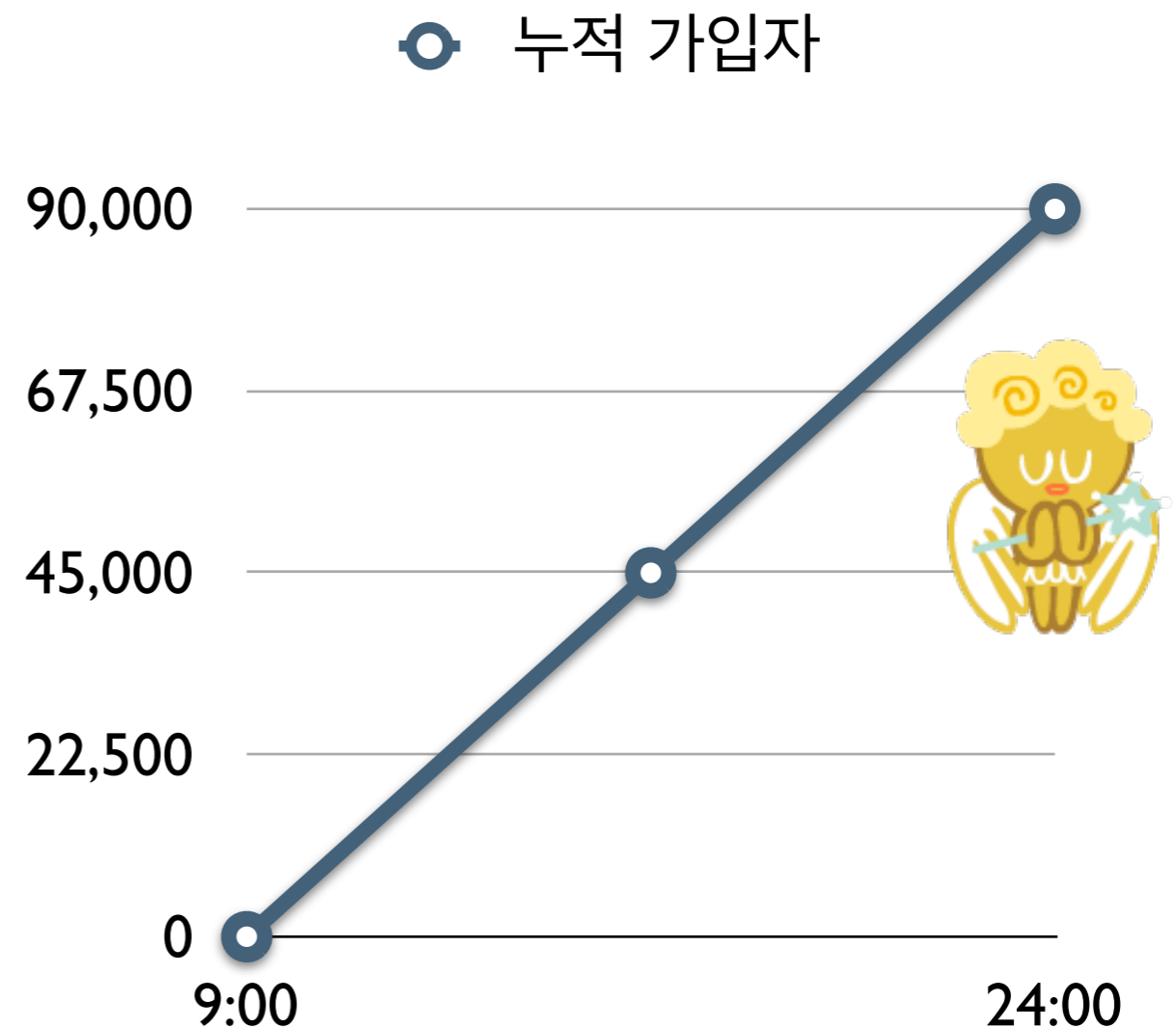
# 출시 직후

DEVSISTERS

# 출시 후 한달, 그 여정

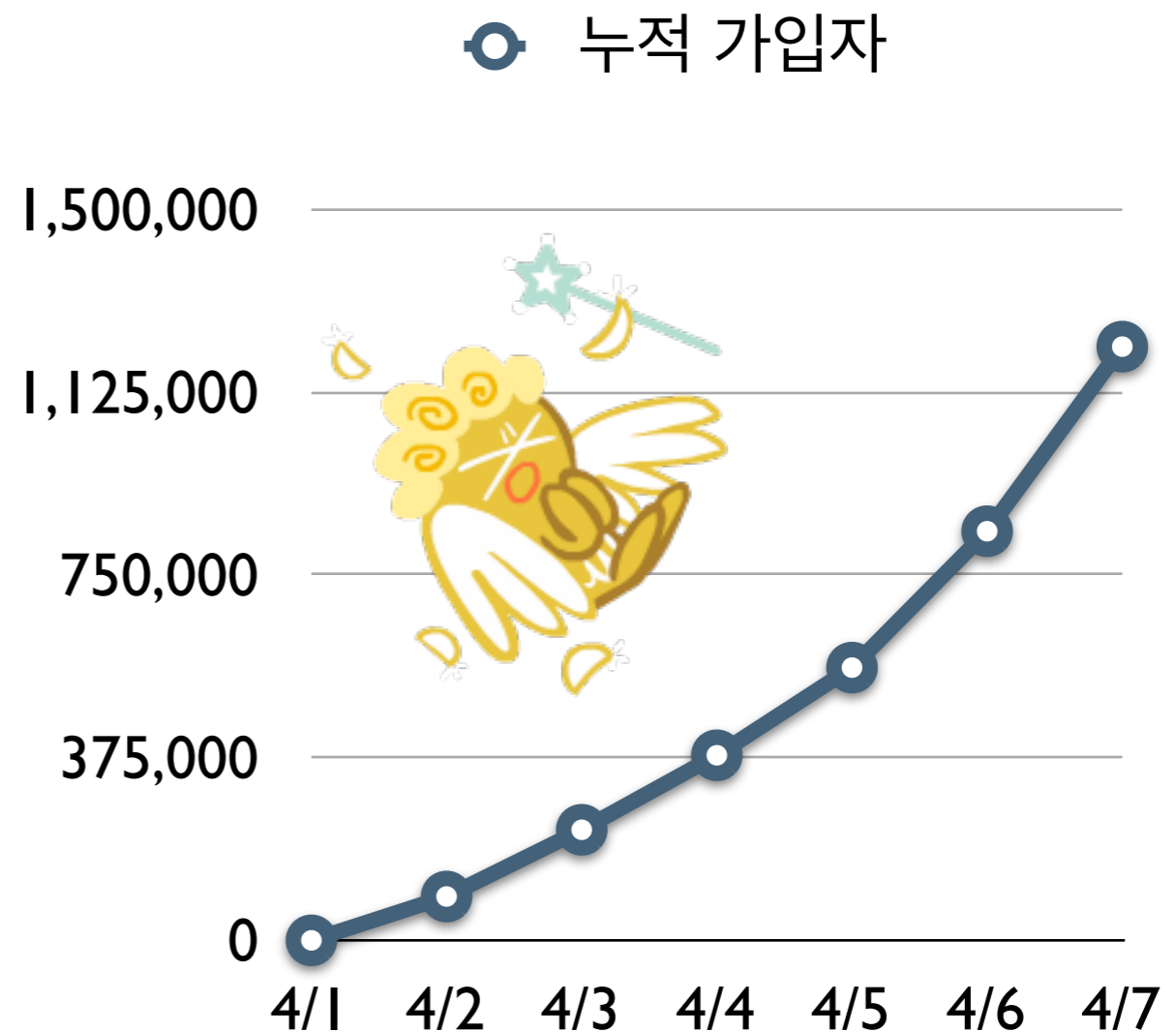
폭발적인 성장과 서버 장애

# 출시 직후의 서버 상황

DEVSISTERS

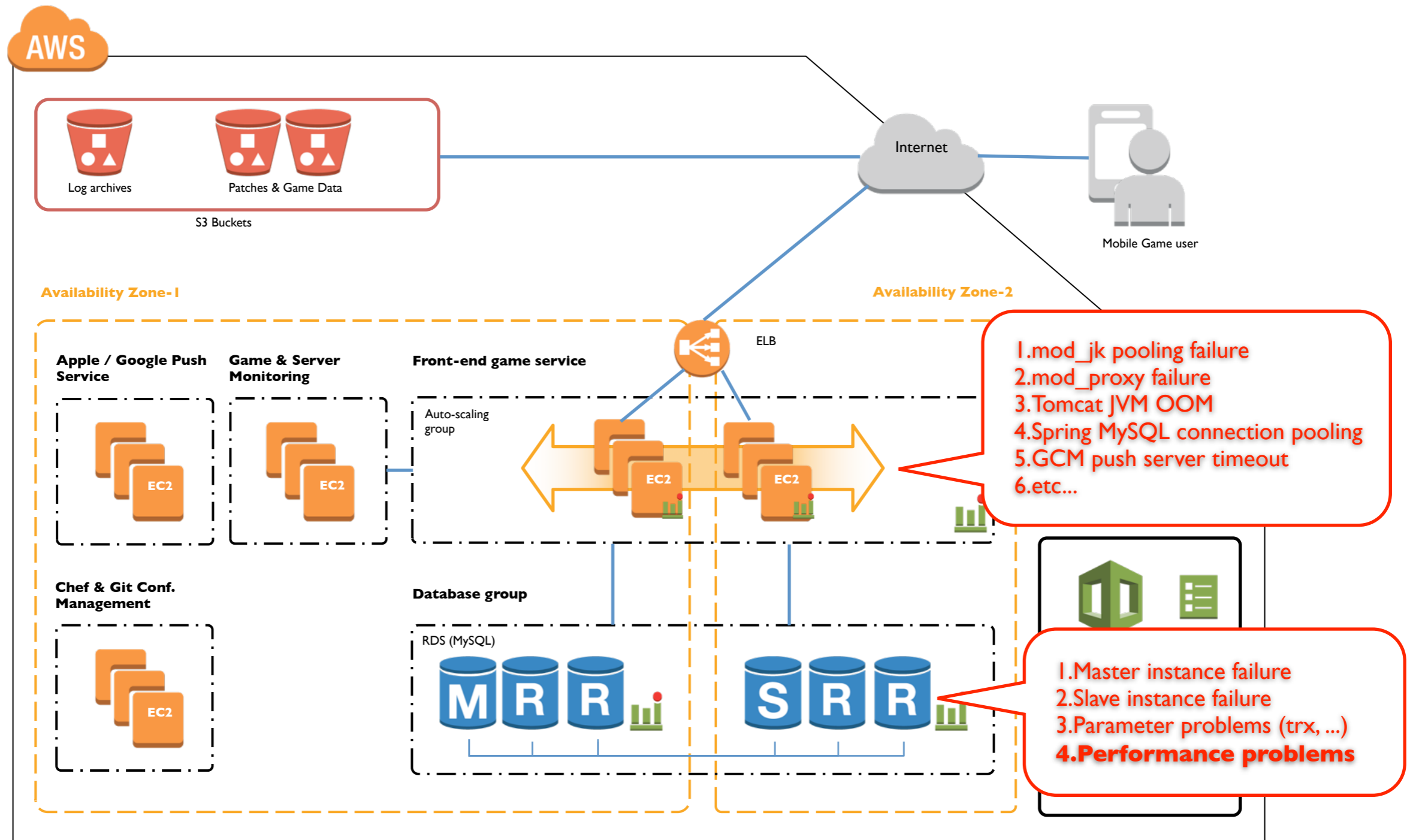
- 미미했던 시작, 얼마나 성장할까?
  - 별다른 마케팅 없이 시작
  - 카카오톡 테마가 유일
- 첫날(4/2) 가입자 9만명!
  - 서버 부하는 버틸만
- 내일도 오늘만큼만 가입해다오!

# 출시 후 일주일

DEVSISTERS

- 산술증가가 아닌 기하급수적 증가
    - DAU도 마찬가지로 증가
- 4일째부터 주기적 서버 장애
    - 정말 다양한 원인
- 6일만에 120만명 돌파
- Auto Scaling 최대 100대 까지

# 초기 장애 원인

DEVSISTERS

# 초기 장애 회고

DEVSISTERS

- 장애는 위(서버)에서부터 발생하고 문제를 해결해나가면 아래(데이터베이스)쪽으로 문제가 점점 이동한다
- 초기 장애는 비교적 진단 및 고치기가 쉽지만 후반 장애는 그렇지 못함
- AWS가 직접적인 문제인적은 없었다
- AWS Business Support에 가입하라
  - Engineer에 따라서는 AWS외적인 문제 디버깅도 도와준다 (단, 영어)
  - 생소한 클라우드 환경에서 문제 발생시 훌륭한 동반자가 되어줌
  - 장애를 신속하게 판단 및 해결하는데 필수적

# 장애의 핵심적인 원인

DEVSISTERS

- 순수한 데이터베이스 부하
    - 게임 시작 및 게임 플레이 정산 부하로 데이터베이스 쓰기 폭주
    - Checkpoint Age 문제로 쓰루풋 급락이후 장애

- 생명 보내기 기능
    - 매 시간마다 N명의 사용자가 M명의 사용자에게 생명을 **“받고 보내기”**
    - 최대 3*24*N*M의 데이터로 데이터베이스 엔트리 급격하게 증가
- MySQL DELETE문의 테이블 락으로 성능 저하
    - 데이터 삭제를 하지 않으니 데이터가 무한정 쌓임

# 장애 해결을 위한 초기 시도

DEVSISTERS

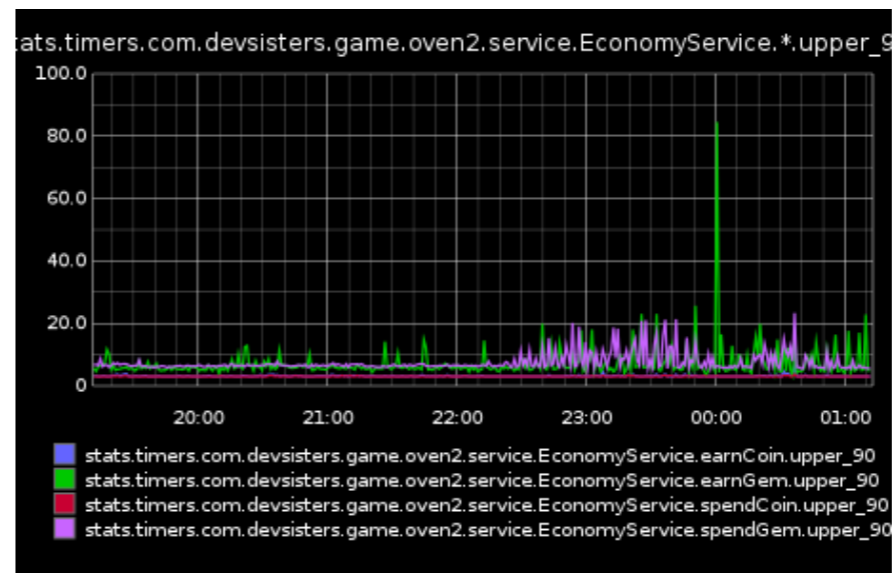
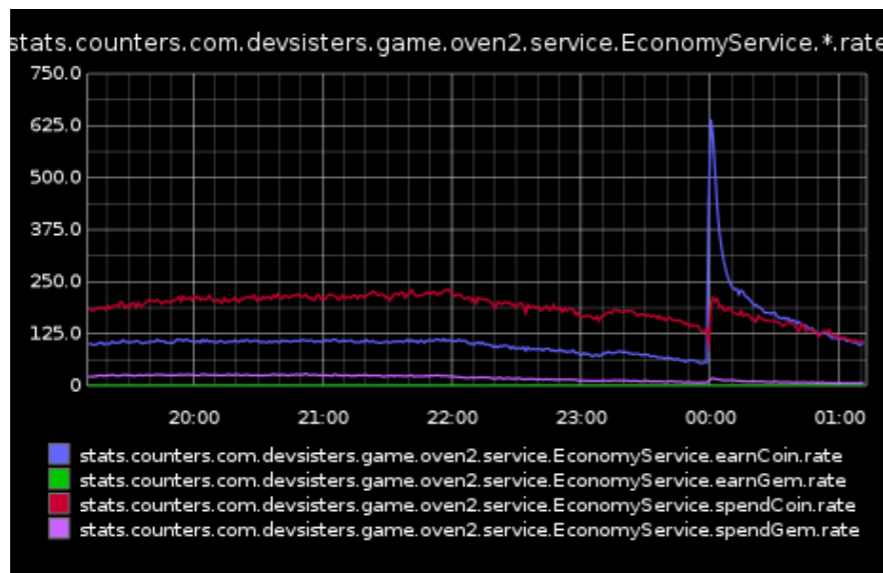
- 데이터베이스 샤딩을 알아보았으나 이미 JOIN이나 데이터 종속이 심함
- 급하게 NoSQL 솔루션을 알아보기 시작
  - Riak: 사용 경험이 있으나 쓰루풋이 낮음 (AP)
  - Couchbase: 성능은 좋아보이나 왠지 상용 버전을 써야할 것 같다 (CP)
  - Redis: 사용하기 쉽고 관리하기 쉽고 쓰루풋도 높다 (250Kops/s)
- 선물포인트, 생명 갯수 등의 주요 핫한 기능을 Redis로 마이그레이션
  - 급한불은 끄고 데이터베이스 부하 저하

# 초기 서버구조 회고(1)

DEVSISTERS

- CloudWatch, SNS, Zabbix, Statsd + Graphite 의 모니터링 조합
- 초기 서버구조 선택 중에 가장 탁월한 선택: 장애/성능문제 판단에 큰 도움
- 메소드 호출빈도, MySQL, 게임 내 경제 (코인 생성/소멸), 결제 모니터링 등

```
@Timed(timingNotes = "")
public void earnCoin(int memberSeq, int amount, String fromWhere)

statsdClient.increment("stat.after_play_earned_coin", earnedCoin);
```

# 초기 서버구조 회고(2)

DEVSISTERS

- EC2 Auto Scaling 으로 API서버의 확장 문제 해결
- **출시 전 AWS의 Instance Limit을 미리 올려놓을것 (기본 20대)**
- 개별 인스턴스의 CPU 사용률은 60% 이하로 유지가 안정적
    - 평균 CPU 사용량 2분동안 60% 이상 → 인스턴스 2대 증가
    - **최대** CPU 사용량 2분동안 80% 이상 → 인스턴스 2대 증가
- 미리 AMI를 빌드하여 인스턴스 부팅부터 서비스 시간을 1분 이하로 줄여놓을 것
- m1.medium, m1.large, m1.xlarge, c1.xlarge 순으로 인스턴스 변경
- 인스턴스는 최대 200대 이하로 유지할 것 (그 이상은 유지관리 비효율적)

# 현 쿠키런 서버 구조

주요 서비스 및 데이터베이스 사용 경험담

# 변경된 사항

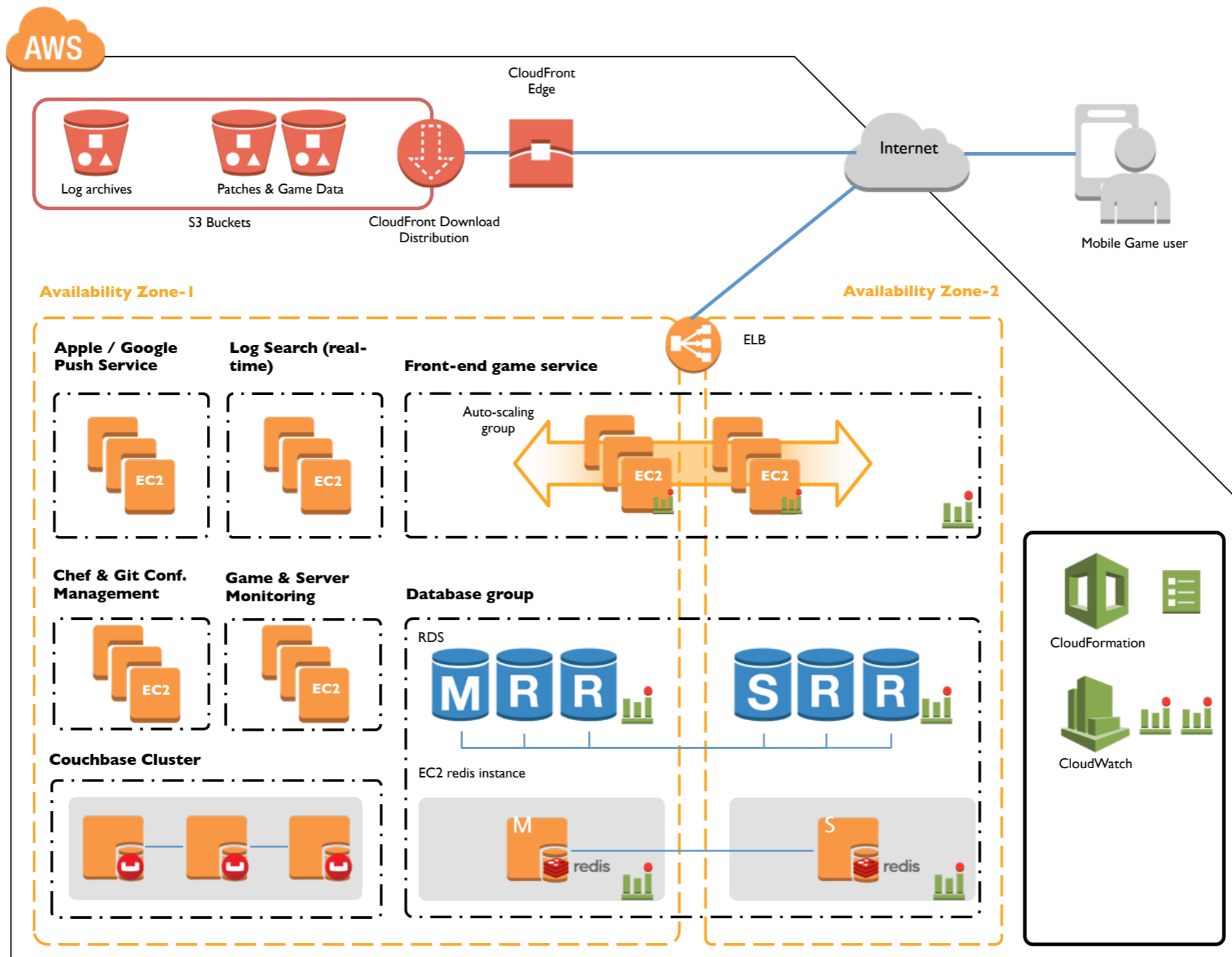
DEVSISTERS

- 실시간 로그 검색/분석 - Logstash, Elasticsearch, Kibana
- Logstash로 로그를 집계하여 S3에 저장 (200GB/day)
- Redis - 생명 개수관리, 점수, 선물포인트, 이벤트정보등의 소셜 정보
- 대용량 푸시 서비스 - Python, Celery
- Couchbase - 생명 우편함 등의 대용량 소셜 정보
- 가장 골치거리였던 생명 우편함을 MySQL에서 Couchbase로 마이그레이션
- Redis의 메모리 한계로 점차 Couchbase로 마이그레이션 중

# 현 서버 구조

DEVSISTERS

# AWS EC2

DEVSISTERS

- 물리서버와 비교하면 유지관리 시간이 거의 소모되지 않음
- AMI, ELB, EBS 등의 조합으로 확장성있는 고가용 서버 구축 가능
- ELB는 정말 무한한 확장 가능, 안정성 문제 단 한번도 없어
- Auto Scaling
    - Scale out 임계치를 잘 설정하는것이 중요 (3개월 정도 튜닝 소요)
    - 장애가 일어난 노드를 알아서 대체함
    - 설정만 잘 해놓으면 더이상 신경쓸일 없음
- 이와같은 환경 설정에는 CloudFormation이 가장 편함

**# instances vs. CPU utilization**

# AWS S3, CloudFront

- S3, AWS 서비스 중에서 단연 최고의 서비스
- 응용 범위가 매우 다양함
- 정적인 파일 저장 및 제공은 S3와 CloudFront CDN으로 모두 해결 가능
- S3의 데이터 내구성 99.999999999%, 가용성 SLA 99.9%
- 하지만 응답속도와 변수를 줄이기 위해 가능한 CDN을 쓰는게 필수적

# AWS RDS

DEVSISTERS

- Replication, Backup 등을 알아서 관리해주는 **매우 편리한** 서비스
  - 온라인 상태에서 데이터베이스 사양 Scale-up 가능
  - 온라인 상태에서 IOPS 증가 가능 (Max 30,000 IOPS)
  - 하지만 사양 변경은 꼭 테스트 후에, 가능하면 새벽 시간에 할 것
- 하드웨어/소프트웨어 장애를 대비하여 **Multi-AZ 옵션 필수적**
- MySQL 5.5 의 경우 Master Failover시 Slave Replica가 깨지는 경우가 발생
  - RDS MySQL 5.6에서 해결

# Redis

DEVSISTERS

- 설치하기 쉽고 쓰기 간편. 쓰기성능도 매우 훌륭
- 데이터구조를 많이 지원하기 때문에 Leaderboard 등 개발 시 매우 편리
- 하지만 MySQL과 마찬가지로 부하 증가시 수동 샤딩등의 방법을 고안해야
- 메모리를 넘어서는 데이터에 대해서는 답이 없음
- 성능 유지를 위해서는 많은 신경을 써야함
  - Fork Latency가 메모리를 많이쓸수록 증가하므로 마스터에서는 하면 안됨
  - 읽기분산을 위해서 디스크 저장용 별도 인스턴스를 유지해야
- 최근 ElastiCache가 지원하기 시작

# AWS와 IDC의 가격 비교

정말 AWS는 저렴한 것일까?

# 비용 측면에서의 AWS

DEVSISTERS

- 단순 서버 vs EC2 비교의 경우 Reserved Instance 를 꼭 고려해야
- 로드밸런서, 방화벽, 운영체제 설치 및 설정은?
- 고가용성 정적 파일 호스팅을 구현하는 비용 (CDN 계약?)
- 데이터베이스 관리는 누가? (Replication, Backup, Monitoring)
- 필요시 바로 서버를 이용할 수 있다는 것 vs 2주 이상 기다려야한다는 것
- Capex vs Opex: 운영비용과 자산비용의 차이
- **혼자 서버를 관리한다면 어떤것을 선택할 것인가?**

# 비용의 진실

DEVSISTERS

- 전통적인 인프라의 관점으로 바라보면 결코 저렴하지 않은 AWS
- 그러나 AWS에서 제공하는 다양한 서비스를 활용한다면 훨씬 비용/시간 효율적
- Pay-as-you-go & Pay-per-only-what-you-use 의 Smart Metering
- 트래픽 증감에 따라 EC2가 Auto Scale; 정확히 사용한 것만 지불하여 비용 절감
- 한 대에서 200대까지 Scale out을 해야한다면 AWS를 사용해야
- EC2 Reserved Instance 요금제로 비용 최적화
- 급격하게 변하는 모바일 게임 시장, AWS는 필연적인 선택
-

# 비용의 진실

DEVSISTERS

- 전통적인 인프라의 관점으로 바라보면 결코 저렴하지 않은 AWS
- 그러나 AWS에서 제공하는 다양한 서비스를 활용한다면 훨씬 비용/시간 효율적
- Pay-as-you-go & Pay-per-only-what-you-use 의 Smart Metering
- 트래픽 증감에 따라 EC2가 Auto Scale; 정확히 사용한 것만 지불하여 비용 절감
- 한 대에서 200대까지 Scale out을 해야한다면 AWS를 사용해야
- EC2 Reserved Instance 요금제로 비용 최적화
- 급격하게 변하는 모바일 게임 시장, AWS는 필연적인 선택
- **저렴하고 확장성있는 서버보다 성공하는 게임을 만드는게 훨씬 어렵다.**
  **재미있는 게임을 만들 수 있도록 기능개발 자체에 투자를 해야.**

# Q&A

# 감사합니다

홍성진 <sungjinhong@devsisters.com>